Statystyczna Analiza Danych
zajęcia laboratoryjne 3

Funkcje

Bioinformatyka II rok

1 Funkcje

- Funkcje nie powinny korzystać ze zmiennych globalnych.
- Funkcje powinny być możliwie krótkie.
- Funkcje w R są traktowane jak zwykłe obiekty.
- Nazwa funkcji nie jest związana z jej definicją, a tylko z nazwą zmiennej, w której funkcja jest zapamiętana.
- Funkcje mogą zwracać wartości. Wynikiem funkcji jest wartość wyznaczona w ostatniej linii ciała funkcji lub można wykorzystać funkcję return().
- Do funkcji można przekazać argumenty, powinny być rozdzielone przecinkami.
- R pozwala na podanie niepełnej, ale jednoznacznej nazwy argumentu. Oznacza to, że w przypadku długiej nazwy argumentów, można podać fragment nazwy, który jednoznacznie identyfikuje argument. Poczytaj o funkcji match.arg().

Przykładowo mamy funkcję przyjmującą 1 argument:

```
1 funkcja <- function(liczba = 10) #wartosc 10 jest wartoscia domyslna
2     liczba * 2 #to samo co: return(liczba*2)
3
4 funkcja() #wywolanie funkcji z argumentami domyslnymi
5 funkcja(licz = 15) #przyklad wywolania bez podania pelnej nazwy argumentu
6 funkcja(5) #wywolanie bez podawania nazwy argumentu
```

Składnia:

```
1 function(argumenty){
2     instrukcje
3 }
4 #jesli w ciele funkcji mamy tylko jedna instrukcje to mozna pominac klamry
5 #argumenty to lista par: nazwa argumentu i jego domyslna wartosc (co jest opcjonalne)
```

Przykład, gdzie przypisujemy funkcję o nazwie function() do zmiennej o nazwie dzielnik (funkcja jest obiektem):

```
dzielnik <- function(liczba){
    for(i in -liczba:liczba){
        if(i == 0){
            next
        }
        if(liczba %% i == 0){
            cat(i, " ")
        }
    }
}
dzielnik(15)
```

Przypisując do zmiennej funkcję, została utworzona zmienna typu funkcyjnego. Chcąc edytować ciało zmiennej można wykorzystać funkcję fix(), przydatne w przypadku używania R z terminala a nie RStudio:

```
fix(nazwa_funkcji)
```

2 Argumenty domyślne

- Definiując funkcję można określić domyślne wartości argumentów. W przypadku wywołania funkcji bez podania argumentów, wykorzystana zostanie wartość domyślna.
- Funkcję można wywołać z dowolną kolejnością argumentów. Jeśli rezygnujemy z kolejności argumentów, należy pamiętać o podaniu nazwy, tak żeby było wiadomo, który argument jest wprowadzany.
- W przypadku, gdy zdefiniowane są argumenty z wartościami domyślnymi to możemy je pominąć w wywoływaniu funkcji.
- W funkcji można wywołać argumenty nadmiarowe, używając wielokropka "...". Argumenty nadmiarowe nie będą wykorzystane w ciele funkcji, ale mogą być przekazane dalej, w wywołaniach innych funkcji.

Przykład - różne sposoby wywołania funkcji:

```
liczby <- c(5,4,3,7,10,2,8)
wyswietl <- function(wektor, od = 2, do = 4){
    wektor[od:do]
}
wyswietl(liczby) #wywolanie z 1 okreslonym argumentem i pozostalymi domyslnymi
wyswietl(liczby, 6) #wywolanie z 2 argumentami (wektor i od)
wyswietl(liczby, do = 6) #wywolanie z 2 argumentami (wektor i do)
wyswietl(liczby, , 6) #wywolanie z pominieciem argumentu (od)
wyswietl(do = 6, wektor = liczby) #wywolanie ze zmieniona kolejnoscia
```

Przykład poniżej dotyczy funkcji z nadmiarową ilością argumentów. Funkcja ta liczy średnią dla podanego wektora i zaokrągla wynik do liczby całkowitej, przy wywołaniu funkcji z domyślnymi wartościami argumentów. Dodanie wielokropka (...) jako argumentu dla funkcji liczącej średnią, pozwala na przekazanie dodatkowego argumentu w wywołaniach innej funkcji, w tym przypadku funkcji round().

```
wektor <- c(9.9, 3.46, 29.2, 5.2, 163, 87, 11.45)
srednia <- function(wektor, ...){
    round(mean(wektor), ...)
}

srednia(wektor) #wywolanie z domyslnymi wartosciami
srednia(wektor, digits=2) #wywolanie z dodatkowym argumentem dla funkcji round()
#digits dla round() okresla liczbe miejsc dziesietnych
```

Kolejny przykład funkcji z nadmiarową ilością argumentów, gdzie pozwalamy na podanie dodatkowych argumentów dla funkcji plot():

```
w <- c(1:10)
rysuj <- function(w, ...){
    plot(w, type = "p", ...) #punktowy typ wykresu
}
rysuj(w, lwd = 3, col = "red") #argumenty dotyczace grubosci i koloru punktow
```

3 Funkcje anonimowe

Funkcje, które jako argument przyjmują inne funkcje. Wyróżniamy następujące sposoby tworzenia takich funkcji:

- Zdefiniowanie funkcji i przypisanie jej do zmiennej, a następnie przekazanie jej jako argument.
- Przekazanie ciała funkcji bezpośrednio jako argument, czyli utworzenie funkcji anonimowej.

Funkcja anonimowa jest to funkcja, która nie jest przypisana do żadnej zmiennej (nie posiada nazwy).

Przykład - argumentem funkcji jest inna funkcja:

```
nazwaFunkcji <- function(x){
    x^2
}
sapply(c(2,4,6), nazwaFunkcji)
```

Przykład - argumentem funkcji jest inna funkcja:

```
sapply(c(2,4,6), function(x) x^2)
```

Sprawdź przy użyciu help jak działa funkcja sapply.

4 Przeciążanie funkcji

- Definiowanie funkcji polimorficznych czyli mechanizm przeciążania funkcji.
- Przeciążanie funkcji oznacza, że funkcja dla określonych argumentów zachowuje się w określony sposób.
- Nie każdą funkcję można przeciążyć.
- Często przeciążanymi funkcjami są: plot(), anova(), print() itd.
- Można sprawdzić czy zadeklarowane są jakieś przeciążone wersje funkcji lub czy istnieją przeciążone funkcje dla jakiej klasy - wykorzystując funkcję methods().
- Mechanizm przeciążania funkcji pozwala tworzyć obiekty określonej klasy i przeciążać dla nich podstawowe funkcje.

Przykładowo funkcja print() wykonuje wiele złożonych czynności, wyświetla wektory, ramki danych, listy i wykonuje to w inny sposób. Funkcja print() jest funkcją przeciążoną. Wykorzystaj funkcję apropos() żeby zobaczyć jak wiele funkcja print() ma do zaoferowania: apropos("print."). W wynikach można znaleźć m.in. funkcję "print.data.frame":

```
dane <- data.frame(col1 = 1:4, col2 = 4:1)
print.data.frame(dane)
print(dane) #wynik jest taki sam jak dla print.data.frame(dane)
```

Do oznaczenia, że dana funkcja jest przeciążona służy funkcja UseMethod(), dzięki temu R wie, że w zależności od typu obiektu może wykonać konkretne działanie. Każda funkcja, która wywołuje UseMethod() nazywana jest funkcją generyczną, a jej zadaniem jest przekazanie działania do innej funkcji, której nazwa ma konkretną postać: nazwaFunkcjiGenerycznej.nazwaKlasy(), w tym przypadku print.data.frame(). Dlatego nie jest konieczne wywołanie funkcji print z nazwą typu obiektu jak w przykładzie powyżej (print.data.frame()), wystarczy wywołać samą funkcję print(), a R sprawdzi wszystkie dostępne funkcje dla konkretnych obiektów. Jeśli funkcja nie określa działania dla konkretnego typu obiektu to zazwyczaj wykonuje się działanie domyślne, można spróbować wyświetlić "dane" w sposób domyślny:

```
print.default(dane)
```

Ważne! - nie każda funkcja może być przeciążona. Jeśli chcemy, żeby R wiedział, że jakaś funkcja może być przeciążona, należy ją oznaczyć używając funkcji UseMethod().

Przykład - niech funkcja dodajProcent() dodaje % do wartości numerycznych po odpowiednim przeliczeniu oraz niech dodaje % do znaków. Funkcja w zależności od charakteru zmiennych ma zachować się w odpowiedni sposób:

```
#1. oznaczamy funkcje jako przeciazona przez UseMethod()
dodajProcent <- function(x){UseMethod("dodajProcent")}
#2. okreslamy domyslne zachowanie funkcji
dodajProcent.default <- function(x){
  paste("Funkcja dziala dla wektora numerycznego lub wektora znakow")
}
#3. okreslamy zachowanie dla konkretnych klas argumentow
dodajProcent.character <- function(x){paste(x, "%", sep = "")}

dodajProcent.numeric <- function(x){
    procent <- round(x * 100, digits = 2)
    paste(procent, "%", sep = "")
}
#4. wywolanie z roznymi argumentami
dodajProcent(c(0.458, 1.6653, 0.83112))
dodajProcent(c("a", "b", "c"))
dodajProcent(c(T,F))
```

5 Funkcje a zasięg

Operatory przypisania wartości do zmiennej, które zostały do tej pory poznane: <−, =. Wyróżnić można jeszcze inny operator przypisania: <<−. Różnica jest taka, że <−, = przypisują wartość zmiennej o lokalnym zasięgu. Zatem użycie któregoś z wymienionych operatorów funkcji, zmienia wartość lokalnie w funkcji. W przypadku, operatora <<− jest to przypisanie wartości zmiennej o globalnym zasięgu, co oznacza, że zmiany są widoczne poza funkcją.

Przykład - klasyczny przykład zamiany wartości, zwróć uwagę że w funkcji do zmiennej b przypisano wartość o zasięgu globalnym:

```
a <- 2
b <- 3
swap <- function(){
  tmp <- a
  a <- b #przypisanie lokalne
  b <<- tmp #przypisanie globalne
  cat(paste("Wartosci wewnatrz funkcji: a = ", a, "b = ", b))
}

```

```
cat(paste("Przed wywolaniem funkcji a =", a, "b =", b))
swap()
cat(paste("Po wywolaniu funkcji a =", a, "b =", b))
```

6 Funkcje a operatory

R umożliwia własne definiowanie operatorów, przez otoczenie znakiem % dowolnego ciągu znaków. Technicznie, operatory są zwykłymi funkcjami z różnicą w sposobie wywołania:

```
"%u%" <- function(a,b){a*b + a+b}
2 %u% 3
```

7 Leniwa i gorliwa ewaluacja

- Leniwa ewaluacja: dotyczy wszystkich funkcji w R. Wartości argumentów zdefiniowane w miejscu wywołania funkcji nie są wyznaczane dopóki ich wartość nie jest potrzebna.
- Gorliwa ewaluacja: argumenty funkcji są wyznaczane przed jej uruchomieniem (do funkcji trafiają wyniki ewaluacji argumentów).

Przykład leniwej ewaluacji:

```
f <- function(a, b = c^2){
    c <- 1:a
    print(b)
}
#"b" ma przypisana wartosc domyslna, ktora jest wyrazenie c^2
#"c" jest okreslone wewnatrz funkcji,
#w przestrzeni w ktorej jest wywolywana funkcja, zmienna "c" moze nie istniec
```

Gdyby wartościowanie było gorliwe, to R zwróciłby błąd związany z brakiem wartości zmiennej “c”.

```
f(6)
#podano tylko pierwszy argument,
#ale wartosc "b" jest wyznaczona na koncu ciala funkcji,
#w tym momencie jest znana wartosc zmiennej "c"

f(6, 1:5)
#gdy podamy oba argumenty, funkcja dziala zgodnie z oczekiwaniami

f(6, c^2)
#wystapi blad, zmienna "c" w wywolaniu znajduje sie w innej przestrzeni nazw
#niz zmienna "c" wewnatrz funkcji
```

8 Zadania

Rozwiąż poniższe zadania, a utworzony skryptu .R zostaw do okazania pod koniec zajęć.

Zad. 1

Napisz funkcję, która przyjmuje dwa argumenty: pierwszym jest wektor, a drugim liczba całkowita b. Program zwraca tylko te elementy wektora, które dzielą się bez reszty przez b. Wywołaj funkcję z przykładowymi danymi.

Zad. 2

Napisz funkcję, która wyznacza pierwiastki równania kwadratowego. Funkcja ta przyjmuje trzy argumenty: a, b i c. Wywołaj funkcję z przykładowymi danymi.

Zad. 3

Napisz funkcję, która jako argument przyjmuje wektor liczb i sortuje te liczby wykorzystując funkcję sort(). Następnie w funkcji wyświetl pierwsze 3 liczby (czyli 3 najmniejsze liczby z wektora). Wywołaj funkcję z przykładowymi danymi.

Zad. 4

Napisz funkcję, która jako argument przyjmuje wektor liczb. Funkcja ma zwrócić w wyniku 3 elementy najmniejsze i 3 elementy największe. Funkcja powinna obsłużyć przypadek, w którym podany wektor zawiera mniej niż 3 elementy. W takim przypadku funkcja powinna wyświetlić komunikat, że wektor jest za krótki. Wywołaj funkcję z przykładowymi danymi.

Zad. 5 - dla chętnych

Napisz funkcję przeciążoną, która przyjmuje 2 argumenty takiego samego typu i w zależności od otrzymanego typu wykona inne instrukcje. Dla dwóch argumentów znakowych np. A i B, funkcja wyświetla: $A \cdot B = AB$, dla dwóch argumentów numerycznych funkcja wykona mnożenie i wyświetli wynik w odpowiedniej postaci, przykładowo dla 2 i 3 otrzymany komunikat powinien wyglądać następująco: $2 \cdot 3 = 6$. Dla pozostałych typów powinna uruchomić się funkcja domyślna, która wyświetli informację, że funkcja działa tylko i wyłącznie dla danych numerycznych i znakowych.